ラダー・ロジック記述を Elixir 言語プログラムに変換する手法についての考察

菊池 豊[1,a)]

概要：工業制御で用いられる PLC はラダー・ロジックで記述される。しかしながら現状の PLC はクラウド技術との親和性が高くない。そこで PLC とクラウドを統合する第 1 歩として、ラダー・ロジックをクラウド技術向きの Elixir 言語へ変換する手法を検討した。

キーワード：Elixir 言語、ラダー・ロジック, 組込システム、PLC、IoT

A study of translation method from ladder logic description into Elixir program

KIKUCHI Yutaka[1,a)]

Abstract: Ladder logic in PLC is popular in FA applications. PLC and cloud technologies should be integrated a united one because of lack of PLC function. This paper proposes an integration process and shows how to translate ladder logic into Elixir program, a first step of the process.

Keywords: Elixir, Ladder logic, Embedded System, PLC, IoT

1. はじめに

著者は地域社会における再生可能エネルギーの利活用、特に小水力発電事業を地域に取り込む活動をしてきた[*1]

地域社会が構築する小水力発電所の出力は、売電目的で系統連系の逆潮流をするタイプで 50kW〜200kW が主流で、より範囲を広げても 10kW〜1000kW 程度である。現在は固定価格買取制度（以下 FIT と略）の元で単体の発電所として採算性のある事業が可能である。

しかし、FIT が定める価格は再生可能エネルギー事業の市場化誘導のため市場価格より高く設定されている。このため、FIT に定める契約期間の 20 年が終了した後には売電価格が激しく下落すると予想されている。また、再生可能エネルギーの先進的な導入が進んでいる欧州の状況を鑑みると、近い将来にエネルギーを産むこと自体より、需給バランスの変化に追従できる電源の方が付加価値が大きくなると予想される。

このため、地域の小水力発電事業は採算性を維持するのが困難になり、他の手法が必要になってくる。例えば、仮想発電所 (VPP: Vritual Power Plant) がその一つの解であり、多くの小さな発電所の集合で、仮想的に大きな発電量や変動能力をもたせることが可能になる。

多くの物理的な発電所を一つの仮想的な発電所として扱うにはクラウド技術が有望である。これは一つ々々の発電所がエッジノードとなり、クラウド上に置かれた制御機能に従って発電出力を制御するものである（図 1)。

しかしながら、発電所は主に PLC (Programmable Logic Controller) によって電力制御が行われている。PLC は元々リレーの置き換えのためのコンピュータであり、単体で動くことが前提であった。このためインターネットに接続して他のシステムと接続するような概念には乏しい。インターネット上のクラウドに接続してエッジノードとしてクラウドの一部を構成するような機能を持ち合わせていない。

これは、小水力発電システムに特有の課題ではなく、これからの工業制御技術のありかたを考える場合に、PLC がクラウドのエッジノード機能を持つことは必須であると考える。このためには PLC のクラウド化等の技術的な進

[1] 高知工科大学
185 Miyanokuchi, Tosayamadacho, Kami-shi, Kochi 782–8502, Japan
[a)] kikuchi.yutaka@kochi-tech.ac.jp

[*1] 小水力発電による地域活性化, 第 3 回スマートエネルギーマネジメントシンポジウム, (2018-11-26)
https://www.slideshare.net/secret/1eWhBDhSlBMOU4

ⓒ 2019 Information Processing Society of Japan

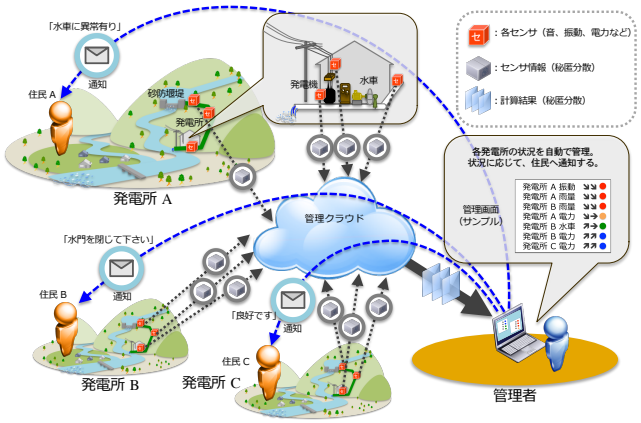

図 1　PLC がクラウドエッジであるシステムの概念
Fig. 1　Concept of systems whose PLCs are the edges of a Cloud

展の他、オープンなコミュニティの醸成などの社会的な環境の進展も必須である。現状では、通称「盤」と呼ばれる PLC 技術とそのエンジニアコミュニティと、Web やクラウドの技術とそのエンジニアコミュニティとは隔絶されている。これを統合した技術としてコミュニティの融合を行う必要がある [1][2]。

この最初のステップとして、PLC のプログラミング言語であるラダー・ロジックをプログラミング言語に変換する手法を確立したいと考えている。クラウド側のプログラミング言語として、後で述べる理由により Elixir 言語を採用した。本稿では、ラダー・ロジックを Elixir 言語に変換して、PLC をクラウドのエッジノードとするための道筋と手法について現状の取り組みと課題を述べる。

2.　動作シナリオ

多数の工場・発電所の装置がエッジノードであり、それらがインターネットで接続されて全体としてクラウドを構成し、一つの機能を実現することを考える。これを実現するために以下の検討が必要である。

- 工業利用環境で稼働するエッジノードハードウェア
- 複数のエッジノードが通信によって相互接続する機能
- 広域自律分散環境において全体として動作する機能

理想的には、クラウドとしての動作を意識しなくてもエッジノードでプログラミングできることが望ましい。

2.1　エッジノードハードウェア

エッジノードとして工業用には現在 PLC を使うのが主流である。しかしながら商用の PLC は OS・ライブラリや開発環境がメーカに依存しており、クラウドに対応させるようなオープンなテクノロジーではない。

一方で Raspberry Pi や Arduino 等の安価な IoT 用デバイス（以下 IoT デバイスと略）が出てきている。これらは OS・ライブラリや開発環境がオープンなコミュニティで開発・公開されており、新しい機能を用意に追加できる他、インターネット上のエッジノードとして使うのが容易である。しかしながら、動作温度・湿度が狭く、振動（加速度）に対する耐性も不明瞭である。このため機械を制御する装置として産業界で受け入れられるには難がある。

以上の機能が同時に提供できるようなエッジノードデバイスに対する要求は以下である。

- 動作する温度湿度や加速度等の設置環境に耐えられる PLC 試験標準 JIS B3502 をクリアできることなど
- 対応する I/O デバイスが多く、新規のデバイスが容易に接続できる
- OS は RTOS であり、ソフトリアルタイム・ハードリアルタイムで動作可能である
- OS・ライブラリ・開発環境等がオープンである
- エンジニアの学習コストが低く、また習得した技術のコミュニティでの還流が期待できる

これらを実現するために PLC 装置自体を開発するプロジェクトを行っており [1]、詳細については別途報告したい。

なお、これらの課題の当面の回避措置としては、PLC と付随する IoT デバイスを接続し、機械制御を PLC で行いネットワーク関係の処理を IoT デバイスで処理するハイブリッド型とでも言うような構成が主流である。これの欠点はハードウェアの費用と開発コストの増加であり、究極的には一体であることが望ましい。

2.2　Elixir 言語

今回、エッジノードを制御する言語として Elixir を採用した。公式ページによると Elixir の特徴は「Elixir は拡張性と保守性の高いアプリケーションを構築するためにデザインされた、動的で関数型のプログラミング言語です。Elixir は、低レイテンシで分散型のフォールトトレラントシステムや、Web や組み込みシステムの領域で成功を収めている、Erlang VM を利用します。」と説明されている[*2]。

Elixir の採用に関して注目したのは以下の特徴である。

- 並行処理に対する記述が容易
- 信頼性が高い
- 組み込み開発で使いやすい
  - 小さな資源で動く・開発できる
  - 新規の入出力デバイスのライブラリが簡単に作れる
  - クロス開発ができる、特に Macintosh で開発ができる
  - ターゲットマシンに OS がなくても（いわゆるベアメタル上でも）稼働する
- 開発のしやすさ
  - ドキュメントが充実していること、特に日本語
  - 安定して言語開発が継続されていること
  - コミュニティが活発なこと
  - 必要な機能が安定版に入ってること

[*2] https://elixir-lang.jp

ⓒ 2019 Information Processing Society of Japan

– クラウドベンダーの SDK が対応していること

特に最初の 2 項目が重要である。まず、I/O を扱う上で並行プログラミングが容易であること、アクタモデルを採用しており低い学習コストで並行プログラミングができることを評価した。さらに、以下の機能より産業用で採用するにあたり十分な信頼性を持ちうると判断した。

- コンパイル言語であり、インタプリタによるスクリプト言語と違い、実行前に一定のエラーを検出できる
- 試験コードを埋込みソースコードで検定が容易
- OTP (Open Telecom Platform) と呼ばれる実行監視システムが言語に組込んである
- 静的型チェックのツールがある （言語自体は動的型付け）

なお、言語選択プロセスの詳細については別途ドキュメントがあるので参照してほしい[*3]。

2.2.1 RTOS 対応

産業用に応用するに当たり実時間動作も重要なポイントになる。Elixir 自体はプロセス[*4]に優先順位の概念があり、絶対優先度と相対優先度が混在した 4 段階のプロセス優先度を持つ[*5]。

しかしながら Elixir が動作する OS は Macintosh・Linux・Windows が主であり、これらの OS が絶対優先度を持たないため、この上で動作する Elixir プログラムもハードリアルタイムのプログラミングができない。

現状では実用上十分なレベルでの応答性能を持つことで、徐々に IoT の分野に応用されてきている。さらに、RTOS 上で Elixir を実行させる処理系として ZEAM[3], GRiSP[*6], AtomVM[*7] 等のプロジェクトがあり、今後に期待したい。

2.3 ラダー・ロジック

PLC のプログラム言語はいくつかあり、そのなかでもラダー・ダイアグラムが代表的である。言語の定義は IEC61131-3 で定められている。ただし言語定義 IEC61131-3 が多くの商品化の後で構成されたこともあり、標準化は浸透しておらず、ほぼ PLC メーカごとの方言となっている。

図 2 は Wikipedia のラダー・ロジックの項目にある自己保持回路のラダー・ロジック図である。図の構造が梯子に似ているためラダーと呼ばれる。ここに示したひとまとまりのリレー回路構成をラングと言い、これも梯子の横棒の表現由来である。

Elixir でこれによる制御を行うためには、以下の手法が考えられる。

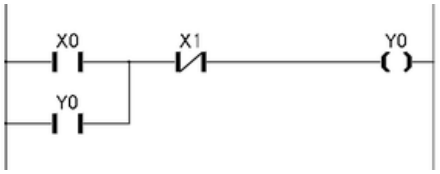

図 2 自己保持回路のラダー・ロジック図
Fig. 2 Ladder logic diagram of Self hold circuit

- ラダー・ロジック図を実行する環境を構成する
- ラダー・ロジック図を Elixir プログラムに変換する
- 同様の動作をする Elixir プログラムを直接記述する

実現容易性を考慮して中間の手法を採用することとした。

3. ラダー・ロジックの Elixir プログラムへの変換方法

前節でラダー・ロジック図を Elixir プログラムに変換する手法を選択した旨を述べた。ただし図による言語からの変換は開発コストが大きいため、最終的には既存の財産であるラダー・ロジック図を変換できることを目標とし、ラダー・ロジック図を表現する式を Elixir プログラムに変換できることを中間的なゴールとした。

3.1 変換モデル

図 2 を論理風に記述したのが以下の式である。

$$Y0' = (X0 \vee Y0) \wedge \neg X1 \tag{1}$$

この 1 つの式が 1 つのラングを表現している。ここで X0 が A 接点の押しボタン、X1 が B 接点の押しボタン、Y0 がリレーの A 接点出力である。Y0' が計算の結果であり、リレーであれば入力すなわちコイル駆動を行う。

3.1.1 素朴なループ

これを最も素朴な方式で Elixir のプログラムにすることを考える。ラングを表す式 (1) を Elixir の式で素朴に表すと以下になる。

```
y0 = (x0 || y0) && -x1
```

この場合、右辺の y0 は式の評価に用いられる。左辺の y0 へは代入ではなく新しい変数領域への値の束縛が行われるため、右辺と左辺の y0 は変数は同じでも互いに干渉しない。左辺の y0 に束縛された値は、次に y0 が評価されるときに用いられる。

これを無限に繰り返すことでリレー回路をシミュレートできる。Elixir のプログラムでは以下のような記述を行う。これは各素子の状態からラングの構造に基づいて新しい状態を再計算することを限りなく繰り返す方式で、通常の PLC での処理手法と同様である。

[*3] Elixir を使うようになった経緯 ～電力システム制御の現場から～, Qiita Elixir Advent Calendar 2018, (2018-12-04), https://qiita.com/kikuyuta/items/96c3f91a850e9274b0e9
[*4] Elixir 言語の実行系である Erlang VM が軽量プロセスを扱い、それをプロセスと呼んでいる。OS のプロセスとは異なる。
[*5] はじめてな Elixir(26) プロセスの優先順位を調べてみる https://qiita.com/kikuyuta/items/f1f7545e283ce838d8dc
[*6] https://grisp.org/
[*7] https://github.com/bettio/AtomVM

ⓒ 2019 Information Processing Society of Japan

```
def loop(y0) do
  {x0, x1} = getinput()  # x0, x1 の状態を取得
  y0 = (x0 || y0) && -x1 # 式を評価して値を y0 に束縛
  loop(y0)               # 末尾再帰による無限ループ
end
```

3.2 素朴な手法の問題点

上で述べた手法でラダー・ロジックに基づく表現を Elixir プログラムとして実行することは可能である。しかし、いくつかの問題をはらんでいる。

(1) 状態変化の伝搬はループ実行に制約を受ける
(2) 全てのラングを一つのループ内に記述する必要がある
(3) 入出力装置を抽象化する際に関数による抽象化に限定される
(4) 状態変化がなくてもループを回し続ける必要がある

1 つ目については、本来とある要素の状態変化は、それが依存している他の要素の状態変化によって起こるはずである。これはループを 1 周する速度の制約を受ける上、意味論が表現と実装とで異なることになる。システムが複雑になった場合、特にクラウド全体で制御するような場合に、意味論の矛盾が起こす歪がシステム全体に波及し、期待する動作をしなかったりデバッグが困難になったりする可能性がある。

2 つ目については、スパゲッティプログラムならぬスパゲッティラングを誘発して、構築や保守を困難にしうる。また、ラダー・ロジックによるプログラムのモジュラ化を妨げ、ソフトウェアパーツとしての再利用を困難にする。

3 つ目については、複雑な装置やネットワーク上の別のシステムの挙動が同一ループ内に表現されることになり、見通しの悪い保守しにくい記述が生成される可能性を示している。

最後の 4 つ目については、エネルギー効率上の問題をはらんでおり、特に資源に乏しい非力なノードもクラウドの要素として考える場合に大きな問題となる。

これらは Elixir プログラムにしたことで起こる問題ではなく、ラダー・ロジック・ダイアグラムの持つ本質的な問題である。素朴な変換手法ではこれらの問題を解決することができない。

3.3 Elixir プロセスによる表現

Elixir 言語を選んだ理由の一つに並列性を自然に表現でき、プログラミングの構成と保守が用意になるという利点を考慮している。これは入出力装置を含む多くの要素をプログラム上で抽象化し、クラウド全体のプログラミングを容易にする可能性とも解釈できる。

上の例ではボタンやリレーをそれぞれ独立した Elixir プロセスとして記述することで抽象化を行い、それらの状態の変化をトリガーとしてシステム全体の評価を行い新しい状態に遷移するようなプログラミングを可能とする。

これの検証を行うため、入出力デバイスを Elixir プロセスとして簡単な電気回路をプログラミングするとどうなるかの記述を行った[*8]。なおこれはラダー・ロジックからの変換ではなく、Elixir プログラムを直接記述してみて検証を行っている。

これによって、Elixir のプロセスは独立性が高く、抽象性や信頼性の面では想定通りの効果を得ることができたと考える。一方で、Elixir プロセスの状態変化からトリガーをかける他のプロセスが何であるかを記載して管理する方法に、これと言った決め手となる手法が見つかっていない。すなわち、ラダー・ロジックのラングの表現をどう Elixir プログラムに変換するのかについてはまだ検討の余地が残されている。

これを実現するのにはいわゆる Pub/Sub モデル (Publisher/Subscriber Model) を用い、Elixir プロセスの状態変化を Pub 側、それによる状態遷移を Sub 側で記述するのが抽象度や保守性の上で望ましいのではないかと考えている。これは Elixir ではいくつかの手法があり、一つは標準ライブラリ Registry を用いる方法、もうひとつはやはり標準ライブラリである GenStage を用いる方法である。

ただしどちらも、状態遷移の結果である Sub の出力が Elixir プロセスの状態を維持している Pub 側にフィードバックされる。このため全体として再帰的な構造を持つことが通常の Pub/Sub モデルと異なる。

4. まとめ

PLC に代表される産業制御システムをクラウド化して、より高度な機能と優れた保守性・信頼性を実現する提案を行った。このために Elixir 言語が備えている機能を用いることができると考えている。変換手法は再帰構造を追加した形の Pub/Sub モデルを検討中であり、この構造が変換手法にどのような影響を与えるのか今後の課題である。

謝辞 Elixir 言語の素敵なコミュニティである fukuoka.ex および kochi.ex の皆さんに、そして高知組み込み会の皆さんに感謝します。

参考文献

[1] 菊池　豊：クラウドと組込との接点を求めて (2019). 日本学術振興会インターネット技術第 163 委員会 RICC-PIoT ワークショップ 2019.
[2] 菊池　豊：工場の制御を Elixir で 〜ラダー・ロジックを実行する〜 (2019). Erlang & Elixir Fest 2019.
[3] 山崎　進, 森　正和, 上野嘉大, 高瀬英希, 小原崇寛：Elixir 処理系 ZEAM ロードマップ，*SWEST20*, No. IS-042 (2018).

[*8] 階段の上でも下でも電灯を点けたり消したりする，Qiita fukuoka.ex Elixir/Phoenix Advent Calendar 2018, (2018-12-13), https://qiita.com/kikuyuta/items/632496c527fcd7c158d5

ⓒ 2019 Information Processing Society of Japan